

Base Porting of Linux Kernel on RISC V Architecture

G Satish Kumar

Index

- | 1. Basics of RISC V ISA
- | 2. RISC V SoC Boards
- | 3. Terminology in SiFive Boards
- | 4. Booting SiFive Kernel
- | 5. Early boot in SiFive Boards
- | 6. setup_arch in SiFive

Index

- 7. SMP init in SiFive Kernel
- 8. Shut Down using SBI
- 9. Traps in SiFive Kernel
- 10. Timer Interrupt in SiFive Kernel
- 11. Paging & MMU in SiFive

Basics of RISC V ISA

- | RV32I -->Base Integer Instruction Set, 32-bit
- | RV32E -->Base Integer Instruction Set (embedded), 32-bit, 16 registers
- | RV64I -->Base Integer Instruction Set, 64-bit
- | RV128I -->Base Integer Instruction Set,128-bit



Basics of RISC V ISA

Register name	Symbolic name	Description (32 integer registers)
x0	Zero	Always zero
x1	ra	Return address
x2	sp	Stack pointer
x3	gp	Global pointer
x4	tp	Thread pointer
x5	t0	Temporary / alternate return address
x6–7	t1–2	Temporary
x8	s0/fp	Saved register / frame pointer
x9	s1	Saved register
x10–11	a0–1	Function argument / return value
x12–17	a2–7	Function argument
x18–27	s2–11	Saved register
x28–31	t3–6	Temporary

RISCV SoC Boards

- | Comerical RISC V SoC boards developed by:
- | 1. SiFive
- | 2. Syntacore
- | 3. Andes Technology
- | 4. Greenwaves Technology
- | 5. Hex Five
- | 6. Western Digital
- | 7. Alibaba Group

RISCV SoC Boards

- | Comerical RISC V SoC boards developed by:
- | 1. SiFive
- | 2. Syntacore
- | 3. Andes Technology
- | 4. Greenwaves Technology
- | 5. Hex Five
- | 6. Western Digital
- | 7. Alibaba Group

Terminology in SiFive Boards

- | RISC V - SiFive Board Terminology:
- | 1. CSR -> Control Status Register Instructions
- | 2. Hart ID -> Hardware Thread ID (CPU ID)
- | 3. Scratch -> Scratch register for supervisor trap handlers
- | 4. Scall -> Make a request to the operating system environment
- | 5. Scause -> Supervisor Cause Register
- | 6. mhartid -> hardware thread id
- |

Terminology in SiFive Boards

- 7. mret -> return from trap in M-mode
- 8. PAGE_OFFSET -> First address of the first page of memory
- 9. stvec -> Supervisor Trap Vector Base Address Register
- 10. sret -> return from traps in s-mode
- 11. sfence.vma -> Synchronize updates memory-management data structures with current execution
- 12. XLEN-1 -> Read only register
- 13. PMP -> Physical Memory Protection

Booting SiFive Kernel

The standard RISC-V privilege model contains four modes:

1. **User mode:** It supports to run user programs.
2. **Supervisor mode:** It supports to run Linux.
3. **Hypervisor mode:** It is currently left unspecified.
4. **Machine mode:** It is the lowest protection mode, and is meant to run the machine-specific firmware that may be microcode on other machines

Booting SiFive Kernel

SiFive boards uses BBL as boot loader.

BBL is developed with support of Device Tree & SBI

1. Device Tree usage: The details of the underlying hardware are described by a device tree,
 - a. specifies the memory map
 - b. configuration of all the harts in the system
 - c. statically allocated devices are attached.
2. SBI usage: Allows to write an interface provided by a lower level of the privilege stack without the hardware complexity of adding a bunch of emulation instructions.

Booting SiFive Kernel

- Bbl's entry point running in **machine mode**.
- It is passed a device tree from the prior boot loader stage, and performs the following steps:
- **One hart** is selected to be the main hart.
- The **other harts are put to sleep** until bbl is ready to transfer control to Linux, at which point they will all be woken up and enter Linux around the same time.

Booting SiFive Kernel

All the other harts are woken up so they can setup their PMP, trap handlers and enter supervisor mode.

The **mhartid CSR** is read so Linux can be passed a unique per-hart identifier

A PMP (Physical Memory Protection) is set up to allow supervisor mode to access all of memory.

Machine mode trap handlers, including a machine mode stack, is set up. bbl's machine mode code needs to handle both unimplemented Instructions and machine-mode interrupts.

Booting SiFive Kernel

- The processor executes a **mret** to jump from machine mode to supervisor mode
- bbl jumps to the start of its payload, which in this case is Linux.

Early boot in SiFive Boards

When Linux boots, it expects the system in below state:

1. a0 contains unique per-hart id, map this hart ID to Linux CPU IDs.
2. a1 contains a pointer to the device tree, represented as a binary flattened device tree
3. Memory is identity mapped.
4. The kernel's ELF image has been loaded, with all the various segments at their addresses.

Early boot in SiFive Boards

Hart Id specification:

- > In Early boot, RISC-V systems boot harts in an arbitrary (Random) order at arbitrary times, while Linux expects a single hart to boot first and then wake up all other harts.
- > This is managed using the "hart lottery" & it is short AMO-based sequence that picks the first hart to boot.
- > The rest of the harts spin, waiting for Linux to boot far enough.

Early boot in SiFive Boards

- | RISC V Linux early boot process:
- | -> A linear mapping of all physical memory is set up, with **PAGE_OFFSET** as the offset.
- | -> **Paging** is enabled
- | -> The **C runtime is set up**, which includes the stack and global pointers.
- | -> A **spin-only trap vector** is set up that catches any errors early in the boot process.
- | -> **start_kernel** is called to enter the standard Linux boot process

setup_arch in SiFive

^λ On RISC-V systems, setup_arch perform the following operations:

- | **Enable the EARLY_PRINTK console**, if the SBI console driver is enabled.
- | The **kernel command line is parsed &** the early arch-specific options
- | are enabled
- | The **device tree's memory map is parsed &** used to find the kernel
- | Image's memory block which is marked as reserved.

setup_arch in SiFive

| Early printk support:
|

```
#ifdef CONFIG_EARLY_PRINTK
static void sbi_console_write(struct console *co, const char *buf,
                             unsigned int n)
{
    int i;

    for (i = 0; i < n; ++i) {
        if (buf[i] == '\n')
            sbi_console_putchar('\r');
        sbi_console_putchar(buf[i]);
    }
}

struct console riscv_sbi_early_console_dev __initdata = {
    .name      = "early",
    .write     = sbi_console_write,
    .flags     = CON_PRINTBUFFER | CON_BOOT | CON_ANYTIME,
    .index     = -1
};
#endif
```

setup_arch in SiFive

| Memory Normal zone init:
|

```
static void __init zone_sizes_init(void)
{
    unsigned long max_zone_pfns[MAX_NR_ZONES] = { 0, };

#ifdef CONFIG_ZONE_DMA32
    max_zone_pfns[ZONE_DMA32] = PFN_DOWN(min(4UL * SZ_1G, max_low_pfn));
#endif
    max_zone_pfns[ZONE_NORMAL] = max_low_pfn;

    free_area_init_nodes(max_zone_pfns);
}
```

setup_arch in SiFive

- Hwcap decides which ISA it should take

```
void riscv_fill_hwcap(void)
{
    struct device_node *node;
    const char *isa;
    size_t i;
    static unsigned long isa2hwcap[256] = {0};

    isa2hwcap['i'] = isa2hwcap['I'] = COMPAT_HWCAP_ISA_I;
    isa2hwcap['m'] = isa2hwcap['M'] = COMPAT_HWCAP_ISA_M;
    isa2hwcap['a'] = isa2hwcap['A'] = COMPAT_HWCAP_ISA_A;
    isa2hwcap['f'] = isa2hwcap['F'] = COMPAT_HWCAP_ISA_F;
    isa2hwcap['d'] = isa2hwcap['D'] = COMPAT_HWCAP_ISA_D;
    isa2hwcap['c'] = isa2hwcap['C'] = COMPAT_HWCAP_ISA_C;

    elf_hwcap = 0;

    /*
     * We don't support running Linux on heterogeneous ISA systems. For
     * now, we just check the ISA of the first processor.
     */
    node = of_find_node_by_type(NULL, "cpu");
    if (!node) {
        pr_warning("Unable to find \"cpu\" devicetree entry");
        return;
    }

    if (of_property_read_string(node, "riscv,isa", &isa)) {
        pr_warning("Unable to find \"riscv,isa\" devicetree entry");
    }
}
```

SMP init in SiFive Linux

```
| Hart lottery procedure:  
| task_struct with kernel's tp (thread pointer) variable and a stack.  
| Assembly Code:  
| .Lsecondary_start:  
|     li a1, CONFIG_NR_CPUS  
|     bgeu a0, a1, .Lsecondary_park  
|     /* Set trap vector to spin forever to help debug */  
|     la a3, .Lsecondary_park  
|     csrw stvec, a3  
|     slli a3, a0, LGREG  
|     la a1, __cpu_up_stack_pointer  
|     la a2, __cpu_up_task_pointer  
|     add a1, a3, a1  
|     add a2, a3, a2
```

SMP init in SiFive Linux

```
λ  /*      This hart didn't win the lottery, so we wait for the winning hart to
      * get far enough along the boot process that it should continue. */
λ  Lwait_for_cpu_up:
      REG_L sp, (a1)
      REG_L tp, (a2)
      beqz sp, .Lwait_for_cpu_up
      beqz tp, .Lwait_for_cpu_up
      fence
      /* Enable virtual memory and relocate to virtual address */
      call relocate
      tail smp_callin
λ  This leaves the __cpu_up function, which boots a target hart by ID,
λ  also to be fairly simple:
```


SMP init in SiFive Linux

```
| Int __cpu_up(unsigned int cpu, struct task_struct *tidle)
| {
|     tidle->thread_info.cpu = cpu;
|     /* On RISC-V systems, all harts boot on their own accord. Our _start
|      * selects the first hart to boot the kernel and causes the remainder
|      * of the harts to spin in a loop waiting for their stack pointer to be
|      * setup by that main hart. Writing __cpu_up_stack_pointer signals to
|      * the spinning harts that they can continue the boot process.
|      */
|     smp_mb();
|     __cpu_up_stack_pointer[cpu] = task_stack_page(tidle) + THREAD_SIZE;
|     __cpu_up_task_pointer[cpu] = tidle;
|     while (!cpu_online(cpu))
|         cpu_relax();
|     return 0;
| }
```


Shut Down using SBI

- | **sbi_shutdown** is called to inform the **machine-mode code** to terminate.
- | File:
- | riscv-pk/machine/mtrap.c

```
void machine_power_off(void)
{
    sbi_shutdown();
    while (1);
}
```

Traps in SiFive Kernel

Traps on RISC-V Systems:

The RISC-V supervisor specification set by writing **stvec CSR**.

The **only way to transfer control to the kernel** is via this entry point.

Effects of taking a trap are **change the PC**, the **exception PC** and **exception cause CSRs**, and **the privilege mode**.

The way to leave the kernel is by executing the **sret** instruction.

The effect of taking a trap is the privilege mode change & the PC is reset to the exception PC CSR's value.

Traps in SiFive Kernel

- The RISC-V ISA uses **sscratch CSR**.
- This CSR provides a single XLEN-sized save region & all software context switching implementations, use this register whatever extra information is actually required to make the context switch.

Traps in SiFive Kernel

- λ **handle_exception**, the Trap Entry Point
- λ Context switching on RISC-V systems are handled by the
- λ supervisor mode software,

the hardware enters the kernel at one single trap entry point and the supervisor-mode software determines how to handle the trap.

There are two categories of traps defined by the RISC-V ISA:

1. **Interrupts:** These are asynchronous. RISC-V defines a software interrupt, a timer interrupt, and an external interrupt.
2. **Exceptions:** These are synchronous. RISC-V defines exceptions to handle instruction, load, store, and AMO access faults; environment calls; illegal instructions; and breakpoints.

Traps in SiFive Kernel

```
ENTRY(handle_exception)
    SAVE_ALL

    /*
     * Set sscratch register to 0, so that if a recursive exception
     * occurs, the exception vector knows it came from the kernel
     */
    csrw sscratch, x0

    /* Load the global pointer */
.option push
.option norelax
    la gp, __global_pointer$
.option pop

    la ra, ret_from_exception
    /*
     * MSB of cause differentiates between
     * interrupts and exceptions
     */
    bge s4, zero, 1f

    /* Handle interrupts */
    move a0, sp /* pt_regs */
    move a1, s4 /* scause */
    tail do_IRQ
```

Traps in SiFive Kernel

- ‡ The trap type is determined by the **scause CSR** upon entry to the trap handler.
- ‡ **Sscratch CSR** take care of saving integer registers to the kernel stack,
- ‡ RISC-V delineates interrupts by setting the high bit in scause,
- ‡ which makes it easy to filter those out and handle them.
- ‡ Most exceptions result from userspace emitting an **scall** instruction
- ‡ to begin a system call, check for that condition and handle the system call using
- ‡ Linux's generic system call handling infrastructure.

Timer Interrupt in SiFive Kernel

- ‡ The SEE determines a timer interrupt occurred with **scause** and enters the supervisor's trap handler, in Linux it is **handle_exception**.
- ‡ Linux calls **do_IRQ** to handle the interrupt.
- ‡ **do_IRQ** calls the **riscv_intc_irq**
(RISC-V interrupt controller driver's interrupt handling function)
- ‡ **riscv_intc_irq** calls **riscv_timer_interrupt**.
- ‡ **riscv_timer_interrupt** looks for the **struct clock_event_device**, which will call to handle the timer interrupt.

Paging & MMU in SiFive

- | Privilege Levels in RISC-V Systems:
- | The RISC-V ISA defines a stack of execution environments.
- | 1. User-mode software executes in an AEE
| (Application Execution Environment).
- | 2. Supervisor-mode software executes in an SEE
| (Supervisor Execution Environment).
- | 3. Hypervisor-mode software executes in an HEE
| (Hypervisor Execution Environment).
- | 4. Machine-mode software executes in an MEE
| (Machine Execution Environment)

Paging & MMU in SiFive

- Pages are 4KB at the leaf node, and it's possible to map large contiguous regions with every level of the page table.
- RV32I-based systems can have up to 34-bit physical addresses with a three level page table.
- RV64I-based systems can have multiple virtual address widths, starting with 39-bit and extending up to 64-bit in increments of 9 bits.
- Mappings must be synchronized via the **sfence.vma** instruction.

Paging & MMU in SiFive

- There are bits for global mappings, supervisor-only, read/write/execute, and accessed/dirty.
- The accessed and dirty bits are strongly ordered with respect to accesses from the same hart.

SiFive Kernel on Qemu

1 <https://risc-v-getting-started-guide.readthedocs.io/en/latest/linux-qemu.html>

```
riscv@localhost ~/riscv-linux $ sudo qemu-system-riscv64 -nographic -machine virt -kernel riscv-pk/b  
uild/bbl -append 'root=/dev/vda ro console=ttyS0' -drive file=busybear-linux/busybear.bin,format=raw  
,id=hd0 -device virtio-blk-device,drive=hd0 -netdev type=tap,script=./ifup,downscript=./ifdown,id=ne  
t0 -device virtio-net-device,netdev=net0
```

```
[ 0.040000] workingset: timestamp_bits=62 max_order=14 bucket_order=0  
[ 0.070000] Block layer SCSI generic (bsg) driver version 0.4 loaded (major 254)  
[ 0.070000] io scheduler noop registered  
[ 0.070000] io scheduler deadline registered  
[ 0.070000] io scheduler cfq registered (default)  
[ 0.070000] io scheduler mq-deadline registered  
[ 0.070000] io scheduler kyber registered  
[ 0.090000] Serial: 8250/16550 driver, 4 ports, IRQ sharing disabled  
[ 0.090000] console [ttyS0] disabled  
[ 0.100000] 10000000.uart: ttyS0 at MMIO 0x10000000 (irq = 10, base_baud = 230400) is a 16550A  
[ 0.100000] console [ttyS0] enabled  
[ 0.100000] console [ttyS0] enabled  
[ 0.100000] bootconsole [early0] disabled  
[ 0.100000] bootconsole [early0] disabled  
[ 0.100000] virtio-blk virtio0: [vda] 524288 512-byte logical blocks (268 MB/256 MiB)  
[ 0.120000] NET: Registered protocol family 10  
[ 0.120000] Segment Routing with IPv6  
[ 0.120000] sit: IPv6, IPv4 and MPLS over IPv4 tunneling driver  
[ 0.130000] EXT4-fs (vda): INFO: recovery required on readonly filesystem  
[ 0.130000] EXT4-fs (vda): write access will be enabled during recovery  
[ 0.160000] EXT4-fs (vda): recovery complete  
[ 0.180000] EXT4-fs (vda): mounted filesystem with ordered data mode. Opts: (null)  
[ 0.190000] VFS: Mounted root (ext4 filesystem) readonly on device 254:0.  
[ 0.200000] Freeing unused kernel memory: 104K  
[ 0.200000] This architecture does not have kernel memory protection.  
[ 0.200000] Run /sbin/init as init process  
[ 0.290000] EXT4-fs (vda): re-mounted. Opts: (null)  
[ 0.470000] IPv6: ADDRCONF(NETDEV_UP): eth0: link is not ready  
Initializing mdev...
```


SiFive Kernel on Qemu

[illegible]

SiFive Kernel on Qemu

```
[ 0.470000] IPv6: ADDRCONF(NETDEV_UP): eth0: link is not ready
Initializing mdev...
[ 1.200000] IPv6: ADDRCONF(NETDEV_CHANGE): eth0: link becomes ready
[ 1.300000] random: dropbear: uninitialized urandom read (32 bytes read)

ucbvax login: ntpd: setting time to 2018-09-13 13:39:29.793853 (offset +1536845967.257232s)
root
Password:
```



```
root@ucbvax:~# uname -a
Linux ucbvax 4.19.0-rc3 #5 Tue Sep 11 09:40:03 CEST 2018 riscv64 GNU/Linux
root@ucbvax:~# pwd
/root
root@ucbvax:~# cd /
root@ucbvax:/# ls
bin          lib          lost+found  sbin        usr
dev          lib64       proc        sys         var
etc          linuxrc     root        tmp

root@ucbvax:/# cd
root@ucbvax:~# cat /proc/cpuinfo
hart      : 0
isa       : rv64imafdcsu
mmu       : sv48

root@ucbvax:~# to
```

Reference

RISC V ISA :

<https://people.eecs.berkeley.edu/~krste/papers/riscv-privileged-v1.9.pdf>

SiFive Base Port:

<https://www.sifive.com/blog/all-aboard-part-6-booting-a-risc-v-linux-kernel>

RISC V getting starting guide with QEMU:

<https://risc-v-getting-started-guide.readthedocs.io/en/latest/linux-qemu.html>

Thank you
&
Questions ?

